代码优化

2020年7月5日 14:44

1. 流图
    a. 基本块(Basic Block)是满足下列条件的最大的连续三地址指令序列
        i. 控制流只能从基本块的第一个指令进入该块。也就是说，没有跳转到基本块中间或末尾指令的转移指令
        ii. 除了基本块最后一个指令，控制流在离开基本块之前不会跳转或者停机
        iii. 输入：三地址指令序列
        iv. 输出：输入序列对应的基本块列表，其中每个指令恰好被分配给一个基本块
        v. 方法：首先，确定指令序列中哪些指令是首指令(leaders)，即某个基本块的第一个指令
            1) 指令序列的第一个三地址指令是一个首指令
            2) 任意一个条件或无条件转移指令的目标指令是一个首指令
            3) 紧跟在一个条件或无条件转移指令之后的指令是一个首指令
        vi. 然后，每个首指令对应的基本块包括了从它自己开始，直到下一个首指令(不含)或者指令序列结尾之间的所有指令
    b. 流图(Flow Graphs)
        i. 每个结点都是基本块
        ii. 从基本块B到基本块C之间有一条边当且仅当基本块C的第一个指令可能紧跟在B的最后一条指令之后执行，此时称B是C的前驱(predecessor)， C是B的后继(successor)
            1) 有一个从B的结尾跳转到C的开头的条件或无条件跳转语句
            2) 按照原来的三地址语句序列中的顺序，C紧跟在之B后，且B的结尾不存在无条件跳转语句
2. 常用代码优化方法
    a. 优化方法分类
        i. 机器无关优化：针对中间代码
        ii. 机器相关优化：针对目标代码
        iii. 局部代码优化：单个基本块范围内的优化
        iv. 全局代码优化：面向多个基本块的优化
    b. 删除公共子表达式
        i. 公共子表达式(common subexpression)：表达式x op y先前已被计算过，并且从先前的计算到现在，x op y中变量的值没有改变，这种x op y的这次出现就是~
        ii. 局部公共子表达式：同一基本块内的公共子表达式
        iii. 全局公共子表达式：跨基本块的公共子表达式
        iv. 注意多个变量下标会改变数组元素时，对数组元素的操作不一定是公共子表达式
    c. 删除无用代码
        i. 无用代码(死代码Dead-Code)：计算结果永远不会被使用的语句
            a) 会因为前面执行过的某些转换而造成
        ii. 复制传播：在复制语句x = y之后尽可能地用y代替x

1) 常用的公共子表达式消除算法和其它一些优化算法会引入一些复制语句(形如x = y的赋值语句)
2) 复制传播给删除无用代码带来机会

d. 常量合并(Constant Folding)：在编译时刻推导出一个表达式的值是常量，使用该常量来替代这个表达式
e. 代码移动(Code Motion)：对循环不变计算，在进入循环之前就对它们求值
   i. 循环不变计算(loop-invariant computation)：不管循环执行多少次都得到相同结果的表达式
   ii. 注意循环不变计算有相对性，可能内层循环不变计算在外层是会变的
f. 强度削弱(Strength Reduction)：用较快的操作代替较慢的操作，如加代替乘
g. 删除归纳变量：在沿着循环运行时，如果有一组归纳变量的值的变化保持步调一致，常常可以将这组变量删除为只剩一个
   i. 归纳变量(Induction Variable)：对于一个变量x，如果存在一个正的或负的常数c使得每次x被赋值时它的值总增加c，那么x就称为~
      1) 如++i，j=i*4，那么j+=4即可，就可以删除I

3. 基本块的优化（局部优化）
   a. 很多重要的局部优化技术首先把一个基本块转换成为一个无环有向图(directed acyclic graph, DAG)
      i. 基本块中的每个语句s都对应一个内部结点N，N的标号是s中的运算符
      ii. 同时还有一个**定值**变量表被关联到N，表示s是在此基本块内最晚对表中变量进行定值的语句。如x=y+z的定值变量是x
      iii. N的子结点是基本块中在s之前、最后一个对s所使用的运算分量进行定值的语句对应的结点。如果s的某个运算分量在基本块内没有在s之前被定值，则这个运算分量对应的子结点就是代表该运算分量初始值的叶结点(为区别起见，叶节点的定值变量表中的变量加上下脚标0)
      iv. 在为语句x=y+z构造结点N的时候，如果x已经在之前某结点M的定值变量表中，则从M的定值变量表中删除变量x
      v. 形如表达式x=y+x如果已有结点对应，则不用修改DAG，只需该结点的标号中增加定值变量x
      vi. 注意数组元素赋值指令不一定是公共子表达式，在构造DAG时要特殊处理，形如a[j]=y，可以创建运算符[]=的结点，子结点为a,j,y；该结点无定制变量表；创建结点后赢杀死所有依赖a的结点；被杀死的结点不能获得定值变量，即不能成为公共子表达式
   b. DAG的作用
      i. 确定哪些变量的值在该基本块中赋值前被引用过：在DAG中创建了叶结点的那些变量
      ii. 确定哪些语句计算的值可以在基本块外被引用：在DAG构造过程中为语句s（该语句为变量x定值）创建的节点N，在DAG构造结束时x仍然是N的定值变量
   c. 活跃变量：其值可能会在以后被使用的变量

i. 从一个DAG上删除所有没有附加活跃变量的根结点(即没有父结点的结点)
ii. 重复应用这样的处理过程就可以基于DAG消除所有无用代码

d. 从DAG重组基本块
i. 对每个具有若干定值变量的节点，构造一个三地址语句来计算其中某个变量的值。倾向于把计算得到的结果赋给一个在基本块出口处活跃的变量(如果没有全局活跃变量的信息作为依据，就要假设所有变量都在基本块出口处活跃，但是不包含编译器为处理表达式而生成的临时变量)
ii. 如果结点有多个附加的活跃变量，就必须引入复制语句，以便给每一个变量都赋予正确的值

4. 数据流分析(data-flow analysis)
a. 数据流分析：一组用来获取程序执行路径上的数据流信息的技术
b. 数据流分析应用：
i. 到达-定值分析(Reaching-Definition Analysis)
ii. 活跃变量分析(Live-Variable Analysis)
iii. 可用表达式分析(Available-Expression Analysis)
c. 在每一种数据流分析应用中，都会把每个程序点和一个数据流值关联起来
d. 语句的数据流模式：
i. IN[s]:语句s之前的数据流值
ii. OUT[s]:语句s之后的数据流值
iii. f_s：语句s的传递函数(transfer function)：一个赋值语句s之前和之后的数据流值的关系
1) 信息沿执行路径前向传播(前向数据流问题) OUT[s] = fs (IN[s])
2) 信息沿执行路径逆向传播(逆向数据流问题) IN[s]= fs (OUT[s])
iv. 基本块中相邻两个语句之间的数据流值的关系：设基本块B由语句s1, s2, … , sn顺序组成，则 IN[si+1]= OUT[si] i=1, 2, … , n-1
e. 基本块的数据流模式
i. IN[B]:紧靠基本块B之前的数据流值
ii. OUT[B]: 紧随基本块B之后的数据流值
iii. 设基本块B由语句s1,s2,…,sn顺序组成，则
1) IN[B] = IN[s1]
2) OUT[B] = OUT[sn]
iv. f_B：基本块B的传递函数
1) 前向数据流问题：OUT[B]= fB(IN[B])，其中fB= fsn·…·fs2·fs1
2) 逆向数据流问题：IN[B]= fB(OUT[B])，其中fB= fs1·fs2·…·fsn

5. 到达定值分析
a. 定值(Definition)：变量x的定值是(可能)**将一个值赋给x的语句**
6. 到达定值方程
7. 活跃变量分析
a. 活跃变量：对于变量x和程序点p，如果在流图中沿着从p开始的**某条路径会引用变量x在p点的值**，则称变量x在点p是活跃(live)的，否则称变量x在点p不活跃(dead)

        i. 某条路径包括循环了一圈回来重新引用，所以for i++的i如果没有被重新定值，在依次循环后保持了值，就是活跃的。同理，如果循环中是常量，但会被引用，就是活跃的
8. 可用表达式分析
    a. 可用表达式：如果从流图的首节点到达程序点p的每条路径都对表达式x op y进行计算，并且从最后一个这样的计算到点p之间没有再次对x或y定值，那么表达式x op y在点p是可用的(available)
        i. 在点p上，x op y已经在之前被计算过，**不需要重新计算**
9. 支配结点和回边
10. 自然循环及其识别
11. 删除全局公共子表达式
12. 代码移动
13. 作用于归纳变量的强度削弱
14. 归纳变量的删除